Lab Assignment - 2

Name :Anish S Ghiya

Registration number : 18BCB0002

Lab Part 1

Dataset: crx.data

Importing dependancies and loading the dataset into a dataframe

In [1]:

```
#Import the required libraries
import pandas as pd
import numpy as np
```

In [2]:

```
#read the file using pandas
df = pd.read_csv('/content/crx.data', header=None)
```

Q1) In the dataset crx.data, name the columns as col1, col2..col16

In [3]:

```
# Column Headers
df.columns = ["Col"+str(s) for s in range(1,17)]
```

For this step we are just changing the name for the columns by using the shorthand representation for list comprehension using a for loop

Q2)Print the last 10 rows of data

In [4]:

```
df.tail(10)
```

Out[4]:

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Col13	Col14	Col15	Col16
680	b	19.50	0.290	u	g	k	v	0.290	f	f	0	f	g	00280	364	-
681	b	27.83	1.000	y	p	d	h	3.000	f	f	0	f	g	00176	537	-
682	b	17.08	3.290	u	g	i	v	0.335	f	f	0	t	g	00140	2	-
683	b	36.42	0.750	y	p	d	v	0.585	f	f	0	f	g	00240	3	-
684	b	40.58	3.290	u	g	m	v	3.500	f	f	0	t	s	00400	0	-
685	b	21.08	10.085	y	p	e	h	1.250	f	f	0	f	g	00260	0	-
686	a	22.67	0.750	u	g	c	v	2.000	f	t	2	t	g	00200	394	-
687	a	25.25	13.500	y	p	ff	ff	2.000	f	t	1	t	g	00200	1	-
688	b	17.92	0.205	u	g	aa	v	0.040	f	f	0	f	g	00280	750	-
689	b	35.00	3.375	u	g	c	h	8.290	f	f	0	t	g	00000	0	-

Q3) Replace the '?' with Not-a-Number

In [5]:

```
# replaces ? with np.nan
df = df.replace('?', np.nan)
```

Q4) Comment on the datatype of variables

In [17]:

```
display(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 16 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Col1    678 non-null    object 
 1   Col2    678 non-null    float64
 2   Col3    690 non-null    float64
 3   Col4    684 non-null    object 
 4   Col5    684 non-null    object 
 5   Col6    681 non-null    object 
 6   Col7    681 non-null    object 
 7   Col8    690 non-null    float64
 8   Col9    690 non-null    object 
 9   Col10   690 non-null    object 
 10  Col11   690 non-null    int64  
 11  Col12   690 non-null    object 
 12  Col13   690 non-null    object 
 13  Col14   677 non-null    object 
 14  Col15   690 non-null    int64  
 15  Col16   690 non-null    object 
dtypes: float64(3), int64(2), object(11)
memory usage: 86.4+ KB

None
```

Alternaltively we can use the dtypes function from the pandas library to get the datatypes of all columns

In [18]:

```
# Alternatively
display(df.dtypes)
```

```
Col1      object
Col2     float64
Col3     float64
Col4      object
Col5      object
Col6      object
Col7      object
Col8     float64
Col9      object
Col10     object
Col11      int64
Col12     object
Col13     object
Col14     object
Col15      int64
Col16     object
dtype: object
```

Form the above we can see that

- **col3, col8 are float datatype**
- **col11, col15 are integer datatype**
- **remaining all are object datatypes**

Errors noticed

Errors noticed

col2, col8, col14 should not be object datatypes but float/int datatypes

To change the datatypes to int and float we can :

`df['Col2'] = df['Col2'].astype(float)`

`df['Col15'] = df['Col15'].astype(int)`

In [11]:

```
df['Col2'] = df['Col2'].astype(float)
df['Col2'].dtypes
```

Out[11]:

```
dtype('float64')
```

In [12]:

```
df['Col15'] = df['Col15'].astype(int)
df['Col15'].dtype
```

Out[12]:

```
dtype('int64')
```

In [21]:

```
df['Col14'] = df['Col14'].astype(float)
df['Col14'].dtype
```

Out[21]:

```
dtype('float64')
```

In [19]:

```
for col in df.columns:
  print(col + " has " + str(df[col].nunique()) + " unqiue values")
```

```
Col1 has 2 unqiue values
Col2 has 349 unqiue values
Col3 has 215 unqiue values
Col4 has 3 unqiue values
Col5 has 3 unqiue values
Col6 has 14 unqiue values
Col7 has 9 unqiue values
Col8 has 132 unqiue values
Col9 has 2 unqiue values
Col10 has 2 unqiue values
Col11 has 23 unqiue values
Col12 has 2 unqiue values
Col13 has 3 unqiue values
Col14 has 170 unqiue values
Col15 has 240 unqiue values
Col16 has 2 unqiue values
```

In [21]:

```
```

Q5)The col16 has + and -, replace them ‘P’and ‘N’respectively

In [25]:

```
#label encoding A16
print(np.array(df['Col16'][0:10]))
df['Col16_le'] = df["Col16"].map({'+':'P','-':'N'})
print(np.array(df['Col16_le'])[0:10])
```

```
['+' '+' '+' '+' '+' '+' '+' '+' '+' '+']
['P' 'P' 'P' 'P' 'P' 'P' 'P' 'P' 'P' 'P']
```

Another alternative is by using the label encoder from the sklearn library

This method especially when the column to be encoded has a lot of values and so mapping each one manually is tough and hence this method can be used

The downfall however is the fat that we cannot set the desired parameters we want as in in this case we wanted + to be mapped to 'P' but then the label was automatically given 0 and 1 values based on order of occurance and this might pose a lot of problems when handling the ordinal data

In [27]:

```
#label encoding using sklearn
import sklearn
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(df['Col16'])
df['Col16_le_sk'] = le.transform(df['Col16'])
print("Classes detected : ", le.classes_)
print(np.array(df['Col16_le_sk'])[0:10])
```

```
Classes detected :  ['+' '-']
[0 0 0 0 0 0 0 0 0 0]
```

Q6) Find and display the number of variables of type 'Object'

In [28]:

```
cat_cols = [c for c in df.columns if df[c].dtypes=="object"]
display(df[cat_cols].head())
```

	Col1	Col4	Col5	Col6	Col7	Col9	Col10	Col12	Col13	Col16	Col16_le
0	b	u	g	w	v	t	t	f	g	+	P
1	a	u	g	q	h	t	t	f	g	+	P
2	a	u	g	q	h	t	f	f	g	+	P
3	b	u	g	w	v	t	t	t	g	+	P
4	b	u	g	w	v	t	f	f	s	+	P

The breakdown of the above code :

Iterating through all the columns which have the datatype set as "object"

In [ ]:

```

```

Lab Part 2

Dataset: loan.csv

Loading the data into a dataframe

In [47]:

```
import pandas as pd
import numpy as np
loan_df = pd.read_csv('/content/loan.csv')
display(loan_df.head())
```

	customer_id	disbursed_amount	interest	market	employment	time_employed	householder	income	date_issued	target

	customer_id	disbursed_amount	interest	market	employment	time_employed	householder	income	date_issued	target
0	0	23201.5	15.4840	C	Teacher	<=5 years	RENT	84600.0	2013-06-11	0
1	1	7425.0	11.2032	B	Accountant	<=5 years	OWNER	102000.0	2014-05-08	0
2	2	11150.0	8.5100	A	Statistician	<=5 years	RENT	69840.0	2013-10-26	0
3	3	7600.0	5.8656	A	Other	<=5 years	RENT	100386.0	2015-08-20	0
4	4	31960.0	18.7392	E	Bus driver	>5 years	RENT	95040.0	2014-07-22	0

Q1) Display the mean of any two variables with continuous values

In [48]:

```
print("Mean of the disbursed_amount is:", loan_df['disbursed_amount'].mean())
print("Mean of the income is:", loan_df['income'].mean())
```

```
Mean of the disbursed_amount is: 14132.2755
Mean of the income is: 71572.28728914999
```

Using the .mean() function from the pandas library to get the mean of the continuos columns

Alternatively we can use the numpy function for sum to get sum of all elements and then divide by the total number of rows

In [49]:

```
print("Mean of the disbursed_amount is:", np.sum(loan_df['disbursed_amount'])/ loan_df.sha
pe[0])
print("Mean of the income is:", np.sum(loan_df['income'])/ loan_df.shape[0])
```

```
Mean of the disbursed_amount is: 14132.2755
Mean of the income is: 71572.28728915
```

Q2) Print the number of discrete variables

Verifying the datatypes and converting the necessary ones to int datatype as discrete values are integers

The code is as follows

`df['col'] = df['col'].astype(int)`

Special note

- ***If there are NAs in the dataset then astype gives error as it does not know how to handle the NA values hence we need to manually fill NAs with an arbitrary value***

In [53]:

```
loan_df.dtypes
```

Out[53]:

```
customer_id                    int64
disbursed_amount             float64
interest                     float64
market                        object
employment                    object
time_employed                 object
householder                   object
income                       float64
date_issued           datetime64[ns]
target                         int64
loan_purpose                  object
loan_purpose                      object
number_open_accounts             float64
date_last_payment         datetime64[ns]
number_credit_lines_12           float64
dtype: object
```

In [56]:

```
loan_df['number_credit_lines_12'] = loan_df['number_credit_lines_12'].fillna(-99)
loan_df['number_credit_lines_12'] = loan_df['number_credit_lines_12'].astype(int)
loan_df['number_open_accounts'] = loan_df['number_open_accounts'].astype(int)
```

Verifying the result of the above conversion

In [57]:

```
loan_df.dtypes
```

Out[57]:

```
customer_id                        int64
disbursed_amount                 float64
interest                         float64
market                            object
employment                        object
time_employed                     object
householder                       object
income                           float64
date_issued               datetime64[ns]
target                             int64
loan_purpose                      object
number_open_accounts               int64
date_last_payment         datetime64[ns]
number_credit_lines_12             int64
dtype: object
```

Iterating through all the columns which have datatype as int64 and saving them into a list and printing out the result

In [65]:

```
discrete_cols = [c for c in loan_df.columns if loan_df[c].dtype=='int64']
print("The number of coluns with discrete values are {} which are {}".format(len(discrete_
cols), discrete_cols))
```

```
The number of coluns with discrete values are 4 which are ['customer_id', 'target', 'numbe
r_open_accounts', 'number_credit_lines_12']
```

The target columns is a categorical variable and hence it will should not be considered into the discrete

Q3)Display the unique values of two variables with discrete values

In [67]:

```
for i in [0,2]:
  print("The unique values in the {} column is {} and the values being {}".format(discret
e_cols[i], loan_df[discrete_cols[i]].nunique(), loan_df[discrete_cols[i]].unique()))
```

```
The unique values in the customer_id column is 10000 and the values being [   0    1    2
...  9997 9998 9999]
The unique values in the number_open_accounts column is 45 and the values being [ 4 13  8
20 14  5  9 18 16 17 12 15  6 10 11  7 21 19 26  2 22 27 23 25
 24 28  3 30 41 32 33 31 29 37 49 34 35 38  1 36 42 47 40 44 43]
```

To find the total number of unique values and the number of rows in which they occur we can use the value_counts function from the pandas library as shown below

In [66]:

```
print("The number rows of each value in the {} column \n".format(catcols[i]))
display(loan_df[discrete_cols[3]].value_counts())
```

```
The number rows of each value in the time_employed column


-99    9762
 2       87
 1       69
 0       31
 3       30
 4       15
 5        4
 6        2
Name: number_credit_lines_12, dtype: int64
```

Since we filled NAs with -99 those are shown also in the table above

Q4) Display the Month with most of loans issued date

Firstly we will check the datatype for the date_issued columns and based on that convertion is applied to convert it from object datatype to datetime feature

In [22]:

```
# checking the datatype for the date_issued columns
print("Before convertion datatype :", loan_df['date_issued'].dtypes)
#converting the datatype to
loan_df['date_issued'] = pd.to_datetime(loan_df['date_issued'])
print("After covertion datatype ", loan_df['date_issued'].dtypes)
```

```
Before convertion datatype : datetime64[ns]
After covertion datatype  datetime64[ns]
```

Now we will extract the month from the date by using the month extracting function from the datetime function set of the datetime library

In [23]:

```
loan_df['date_issued_month'] = loan_df['date_issued'].dt.month
```

In [24]:

```
loan_df['date_issued_month'].value_counts().sort_values(ascending=False)
```

Out[24]:

```
10    1277
7     1066
11    1017
12     882
8      852
4      816
5      749
9      734
1      700
6      700
3      623
2      584
Name: date_issued_month, dtype: int64
```

In [25]:

```
index_val = loan_df['date_issued_month'].value_counts().sort_values(ascending=False).inde
x[0]
print("The total number of entries in the month number {} are {}".format(index_val, #to ge
t the value of index at first position while putting in decending order
                                                                                  loan_df['date_i
ssued_month'].value_counts().sort_values(ascending=False)[index_val])) # to get value pres
```

```
ent at tht index
```

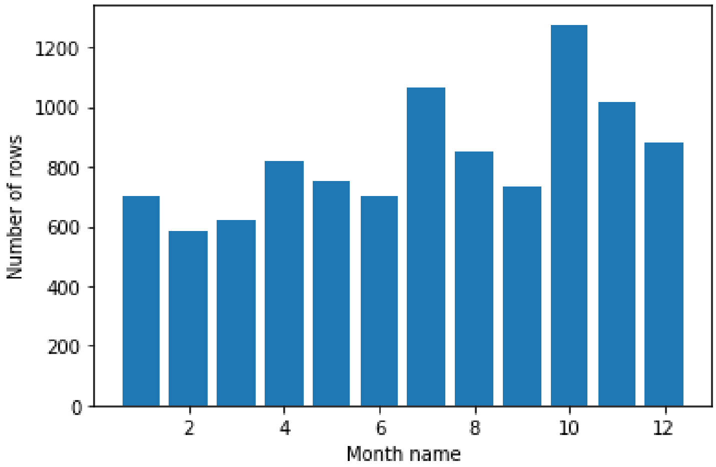
```
The total number of entries in the month number 10 are 1277
```

In [26]:

```
import matplotlib.pyplot as plt
plt.bar(loan_df['date_issued_month'].value_counts().sort_values(ascending=False).index, l
oan_df['date_issued_month'].value_counts().sort_values(ascending=False))
plt.xlabel("Month name")
plt.ylabel("Number of rows")
```

Out[26]:

```
Text(0, 0.5, 'Number of rows')
```



From the table as well as the bar chart the maximum number of entries are present int the month number 10 which has a value of 1261

Q5) Display the count of ‘Teacher’ who are ‘Owners'

In [27]:

```
gpby = loan_df.groupby(['employment', 'householder'],as_index=False)['customer_id'].count
()
gpby[(gpby['employment']=="Teacher") & (gpby["householder"]=="OWNER")]
```

Out[27]:

	employment	householder	customer_id
31	Teacher	OWNER	69

The operations performed here were first grouping by the two columns employement and householder and then generating a condition where employment is 'Teacher' and householder is 'owner'

Q6) Display the ‘Employment’ of customers who mostly ‘Rent'

In [28]:

```
grpby = loan_df.groupby(['householder', 'employment'],as_index=False)['customer_id'].coun
t()
grpby[grpby['householder']=="RENT"].sort_values(by='customer_id', ascending=False)
```

Out[28]:

	householder	employment	customer_id
24	RENT	Civil Servant	371
32	RENT	Teacher	371
23	RENT	Bus driver	360

	householder	employment	customer_id
26	RENT	Nurse	358
25	RENT	Dentist	355
28	RENT	Secretary	355
27	RENT	Other	353
30	RENT	Statistician	342
22	RENT	Accountant	322
31	RENT	Taxi driver	316
29	RENT	Software developer	315

Form the table we can note that the total number of teachers are highest who mostly rent

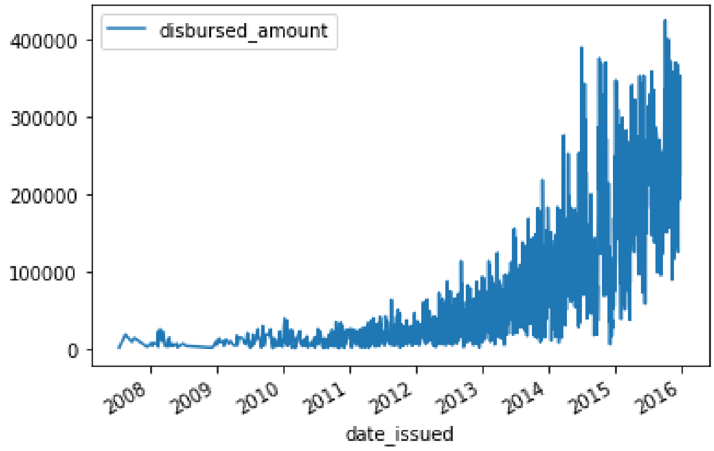
Further analysis

In [32]:

```
loan_df.groupby(['date_issued'],as_index=False)['disbursed_amount'].sum().plot(x='date_iss
ued', y='disbursed_amount')
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f7912e850>
```



From the graph we can observe that the disbursed amount keeps incresing with time. The reason I think is inflation. With the decreasing in the amount we can buy with the same amount of money people start taking bigger loans in order to finance their business.

And so in the future we can see the trend we can see is moving upwards only.

As we can see with the prices of petrol in India where now it is 93 and nearing 100.

Defitinition of Inflation from wikipedia

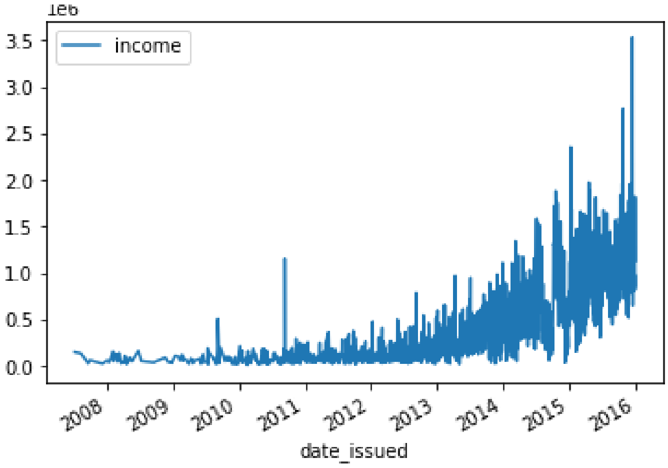
Inflation is the decline of purchasing power of a given currency over time. A quantitative estimate of the rate at which the decline in purchasing power occurs can be reflected in the increase of an average price level of a basket of selected goods and services in an economy over some period of time

In [76]:

```
loan_df.groupby(['date_issued'],as_index=False)['income'].sum().plot(x='date_issued', y='
income')
```

Out[76]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4f78df4d90>
```



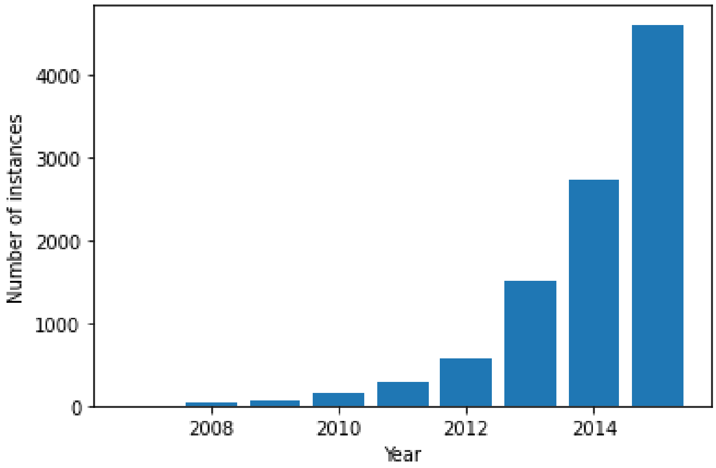
Income has a similar upward trend and this follows the above entioned priciple as well !!

In [93]:

```
import datetime as dt
plt.bar(loan_df['date_issued'].dt.year.value_counts().index, loan_df['date_issued'].dt.year.value_counts())
plt.xlabel("Year")
plt.ylabel("Number of instances")
```

Out[93]:

```
Text(0, 0.5, 'Number of instances')
```



It is clear that the number of instances are far less during 2008 that in 2015

This may be be due to 2 reasons

- **No proper records available from that time**
- **People didn't need so much loan back then because they used to save enough ;P bowadays the concept of savings is slowly dissapearing and hence the increased loan amounts**

In [95]:

```
loan_df['loan_purpose'].value_counts()
```

Out[95]:

```
Debt consolidation    8214
Other                  880
Home improvements      615
Car purchase            90
Health                  87
Holidays                48
Moving home             47
Wedding                 19
Name: loan_purpose, dtype: int64
```

In [100]:

```
loan_df.groupby(['loan_purpose'], as_index=False)[['income', 'disbursed_amount']].mean()
```

Out[100]:

	loan_purpose	income	disbursed_amount
0	Car purchase	72766.650362	8653.897222
1	Debt consolidation	70561.270602	14751.911036
2	Health	63698.271034	8205.712644
3	Holidays	59726.123579	6759.458333
4	Home improvements	87548.138292	13717.299187
5	Moving home	61205.188936	8846.718085
6	Other	71582.128635	10560.726420
7	Wedding	77051.157463	9893.789474

Few facts from the above output :

- **There are people who take loan for a holidays !! Although the amount is least it astonishes me that people would take a loan for a luxury like a holiday trip**
- **The maximum amount of loan taken in terms of mean income is the home loan which make sense as aain it is a luxury and people with higher income would want to do and then repayment is also high !!**

Lastly the loan dataset can be checked for a classification problem where we determine whether the person will be a defaulter or not in terms of whether the person will repay the loan amount or not

Another regression problem can be formulated (with additional data on the amount pending by the person) on how much amount should be samctioned to the person based on profile and nothing greater than that should be sanctioned so that even the people taking the loan are rest assured that they can surely pay back and also not have to pay any kind on penalty

In [ ]:

```

```